

Taller Module Federation

Paso a paso:

1. Crear un directorio y en esa ruta instalar: `npx create-mf-app`
2. Sigues el paso a paso que te va sugiriendo la consola (recuerda que el taller estará orientado en React):

```
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
? Pick the name of your app: appOne
? Project Type: Application
? Port number: 4001
? Framework: react
? Language: typescript
? CSS: Tailwind
? Bundler: Webpack
Your 'appOne' project is ready to go.
```

3. Luego, entra al directorio de “appOne” y allí realiza el respectivo **npm install** y **npm start** desde la misma consola.
4. Luego seguiremos exactamente los mismos pasos (del 1 al 3) y crearemos nuestra segunda app **appTwo**

```
C:\Users\Administrator\Documents\Célula\26-07- Module Federation - FrintJS\MF>npx create-mf-app
? Pick the name of your app: appTwo
? Project Type: Application
? Port number: 4000
? Framework: react
? Language: typescript
? CSS: Tailwind
? Bundler: Webpack
Your 'appTwo' project is ready to go.
```

5. Para este punto, algo así debería verse en el navegador:

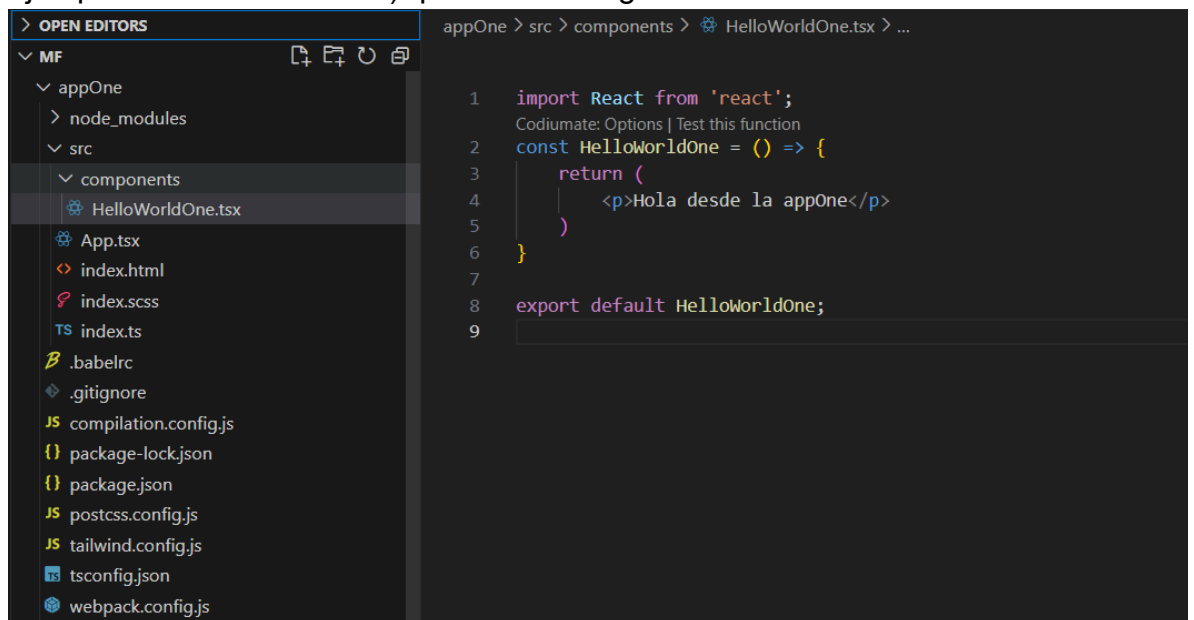


Name: appOne
Framework: react
Language: TypeScript
CSS: Tailwind

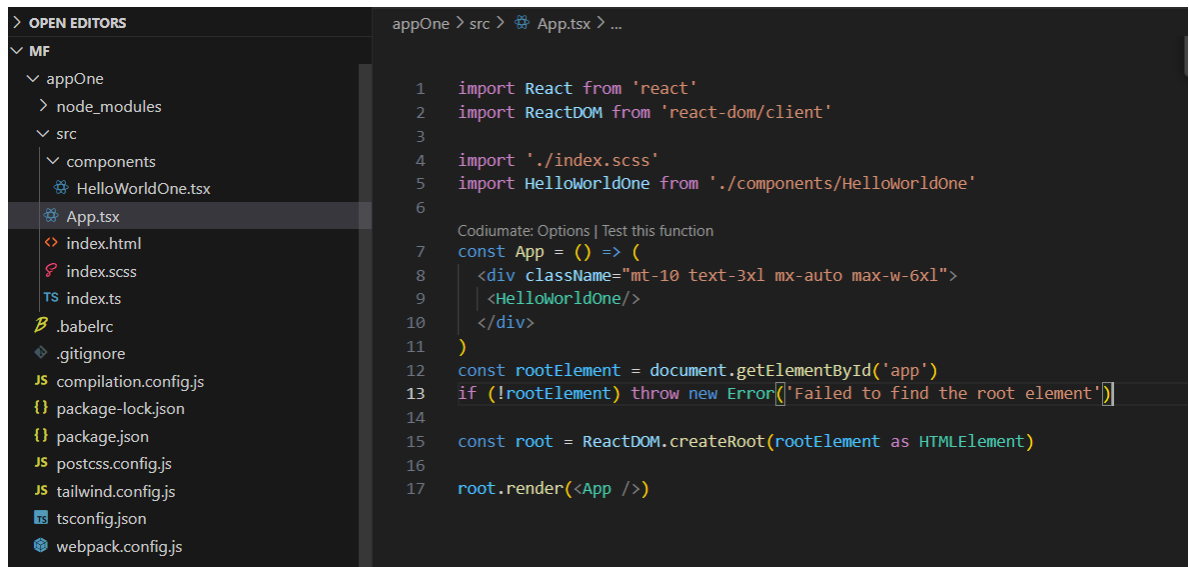


Name: appTwo
Framework: react
Language: TypeScript
CSS: Tailwind

6. Para este punto crearemos dentro de **src** de **appOne**, un componente (el ejemplo es un Hola Mundo) que se vería algo así:

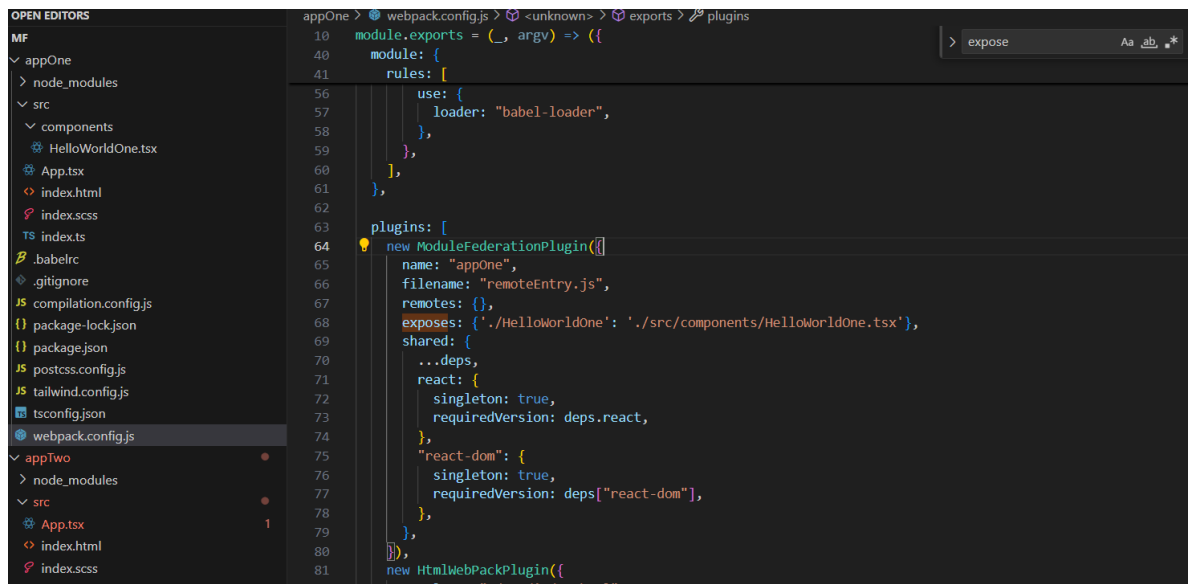


7. Importamos nuestro componente en **App.tsx**:



```
1 import React from 'react'
2 import ReactDOM from 'react-dom/client'
3
4 import './index.scss'
5 import HelloWorldOne from './components/HelloWorldOne'
6
7 Codiumate: Options | Test this function
8 const App = () => (
9   <div className="mt-10 text-3xl mx-auto max-w-6xl">
10     <HelloWorldOne/>
11   </div>
12 )
13
14 const rootElement = document.getElementById('app')
15 if (!rootElement) throw new Error('Failed to find the root element')
16
17 const root = ReactDOM.createRoot(rootElement as HTMLElement)
18 root.render(<App />)
```

8. Una vez llegamos a este punto, abrimos el archivo de **appOne/Webpack.config.js**, y en el objeto de plugins/exposes, “expondremos” el componente que acabamos de crear, de la siguiente manera.



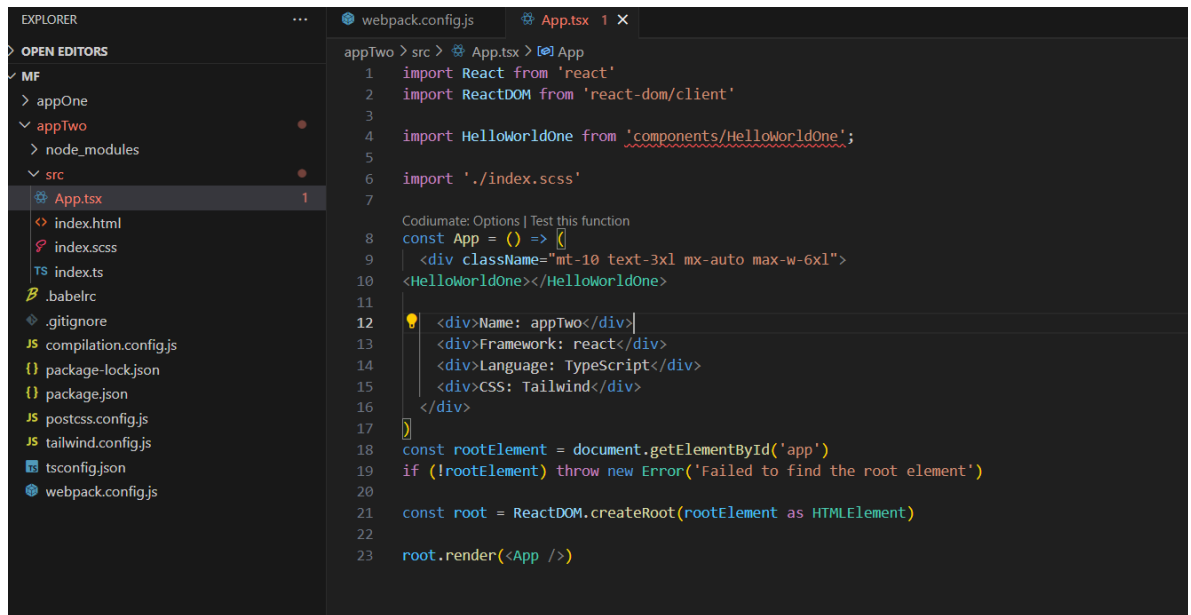
```
10 module.exports = (_, argv) => ({
40   module: {
41     rules: [
56       use: {
57         loader: "babel-loader",
58       },
59     ],
60   },
61 },
62
63 plugins: [
64   new ModuleFederationPlugin({
65     name: "appOne",
66     filename: "remoteEntry.js",
67     remotes: {},
68     exposes: { './HelloWorldOne': './src/components/HelloWorldOne.tsx' },
69     shared: {
70       ...deps,
71       react: {
72         singleton: true,
73         requiredVersion: deps.react,
74       },
75       "react-dom": {
76         singleton: true,
77         requiredVersion: deps["react-dom"],
78       },
79     },
80   }),
81   new HtmlWebpackPlugin({
```

Una vez finalizada la configuración, detenemos la ejecución de **appOne** en la terminal y la volvemos a ejecutar con **npm start**. Allí no se evidenciará cambio alguno, sino que acabamos de exponer nuestro archivo en “remoteEntry.js” y con esto confirmaremos que ya podemos reutilizar nuestro componente en donde queramos. Para efectos de prueba, revisaremos que esté OK y nos dirigimos a esa ruta y traerá algo similar a

[illegible]

-

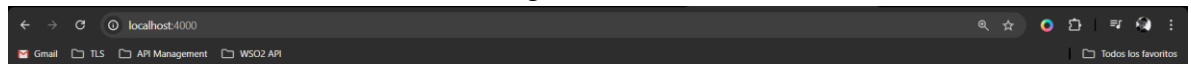
11. Como último paso, solo queda llamar desde nuestra “**App.tsx**” a nuestro componente a integrar como si fuera un componente más de nuestra aplicación:



```
1 import React from 'react'
2 import ReactDOM from 'react-dom/client'
3
4 import HelloWorldOne from 'components/HelloWorldOne';
5
6 import './index.scss'
7
8 const App = () => {
9   <div className="mt-10 text-3xl mx-auto max-w-6xl">
10     <HelloWorldOne></HelloWorldOne>
11   </div>
12   <div>Name: appTwo</div>
13   <div>Framework: react</div>
14   <div>Language: TypeScript</div>
15   <div>CSS: Tailwind</div>
16 </div>
17 }
18
19 const rootElement = document.getElementById('app')
20 if (!rootElement) throw new Error('Failed to find the root element')
21
22 const root = ReactDOM.createRoot(rootElement as HTMLElement)
23
24 root.render(<App />)
```

Algo muy importante, el import siempre debe tener el “alias” del anterior paso + “/” + nombre del componente a llamar.

Con esto, tendremos un resultado algo así:



Nota: Recuerden bajar o detener la ejecución de la app cada vez que modifiquen el archivo de Webpack.config.js para así poder visualizar los cambios en el navegador.